Engenharia Informática

Ramo de Desenvolvimento de Aplicações

Unidade Curricular de **Programação Distribuída**

Ano Curricular de 2020/2021

Trabalho Prático

Meta 1

Realizado por:

Diogo Lima	Nº 2014010835
Gabriel Gomes	Nº 2017018332
Sérgio Soares	Nº 2016014425

Índice

Introdução

No âmbito da unidade curricular de Programação Distribuída, foi proposto a realização de um projecto cliente servidor a simular uma aplicação de chat.

Nesta aplicação será possível utilizadores comunicarem entre si, em privado ou através de canais com vários utilizadores.

Este relatório está dividido em quatro partes sem contar com a introdução. Numa parte inicial será especificada toda a estrutura de classes, seguida de uma breve descrição de cada uma, assim como também uma pequena seção a mencionar o encapsulamento.

Numa segunda parte são documentadas as decisões tomadas ao longo do desenvolvimento do projeto.

De seguida são enumeradas as funcionalidades por implementar.

Por último uma breve conclusão em que se reflete no processo de desenvolvimento.

1 - Classes

No processo de desenvolvimento, foram criadas as classes necessárias para estruturar o trabalho da forma como foi visionado pelos autores.

Inicialmente foram criadas as classes básicas que representavam o User e o servidor. A partir destas foi necessário criar outras classes que representassem os diversos elementos da aplicação como as mensagens e os canais, assim como uma para perfil de utilizador que serviria para passar entre processos.

Foi também criada uma classe para troca de mensagens entre user e servidor de modo a que toda a informação fosse mais facilmente enviada.

Cada elemento do programa tem a sua respetiva classe, para representar o respetivo elemento.

1.1 - Classes principais

A classe principal do lado do servidor, é na **Server** que está a main() por onde vai começar o servidor. É responsável por guardar todos os portos pelos quais se vão efetuar ligações entre servidor e cliente, o serverIP, todas as threads que vão ser necessárias do lado do Servidor (ServerThread, MulticastThread, HeartbeatThread, PingReceiverThread, MulticastSocket) e de guardar num HashMap de chave Integer e de valor UserThreads todas as UserThreads inicializadas, guarda os UserProfile, Channel e PrivateChannel criados em cada respetivo ArrayList do referido objeto e guarda todos os ficheiros num List de File.

É no servidor que se inicializam todas as threads necessárias para o funcionamento do lado do servidor através dos métodos createServerFile();, readServerInfo();, launchHeartbeatThread();, launchServerThread(); e launchMulticastThread();.

Depois de inicializadas as threads, o servidor estabelece a conexão UDP com o cliente, depois de estabelecida esta ligação o Server lança a ServerThread para efetuar a ligação TCP com o cliente.

A classe principal do lado do cliente, é na **User** que está a main() por onde vai começar o utilizador. É responsável por guardar todos os portos e as sockets pelas quais se vão efetuar as ligações entre o cliente e servidor, o UserProfile do utilizador e todas as threads que vão ser necessárias do lado do Utilizador (StethoscopeThread, NotificationThread, FileThread).

É no User que se vai estabelecer a ligação UDP com o servidor, depois da ligação procede-se ao lançamento de todas as threads necessárias para o funcionamento do lado do utilizador através dos métodos launchStethoscopeThread(), launchNotificationThread(); e launchFileThread();.

Depois de inicializadas as threads é enviado para o mainmenu();. É no User que se pede toda a informação sobre a ação a fazer para enviar ao servidor através do UserMessages, e recebe as respostas do servidor através de um objeto Notidfication ou através de uma String.

É também no User que se apresenta o chat, em que a partir do momento em que o utilizador entra no sendMessage(...) notifica-se a NotificationThread para esperar mensagens e, caso seja escolhido enviar um ficheiro, notifica-se a FileThread no método sendFile(...).

Praticamente todas as ações do utilizador são processedas e enviadas para o servidor no User.

1.2 - Encapsulamento

Foram tomadas medidas para que o encapsulamento fosse um elemento bastante robusto do código. O acesso a objetos de outras classes foi limitado, tentando ser o mínimo possível para evitar problemas de manipulação de dados em classes exteriores.

Para propósitos de envio entre Servidor e User foram criadas classes que para representar pedidos feitos, por parte do servidor, com toda a informação necessária já neste embutida.

Todos os métodos das classes são privados e têm o seu getter.

As operações correspondentes a um determinado objeto foram por norma encapsulados dentro desse mesmo objeto. Deste modo torna-se mais simples a manipulação de dados dessa classe.

1.3 - Descrição das Classes

***User*:**

Classe representante do User. Esta engloba tudo o que o User tem incluindo as socket, o seu perfil e as suas threads.Nesta são apresentados os outputs para interação com o utilizador.

Nesta são também enviados todos os pedidos ao servidor diretamente, podendo vir respostas diretas ou por notificação, esta última através de uma thread.

***Server*:**

Classe representante do Server. Esta engloba tudo o que o servidor pode vir a fazer e a lançar. Nesta são apresentados os outputs para interação com o utilizador.

Nesta são também lançadas todas as threads e sockets, seja TCP ou UDP. Devido a um problema a descrever mais abaixo, contém também ArraysLists dos objectos do programa.

***Channel*:**

Classe serializável responsável por guardar os dados referentes a qualquer canal, recebe o seu nome, password e o criador do Canal. Esta guarda todos os utilizadores adicionados num HashSet de String, guarda também as mensagens enviadas no canal num HashMap com chave Integer e com o valor de um objeto Message.

***Message*:**

Classe serializável responsável por guardar cada mensagem enviada em qualquer canal ou canal privado, recebe a mensagem a guardar e o username de quem a enviou.

***Notification*:**

Classe serializável responsável por guardar uma notificação que vai ser enviada do servidor para o cliente para informar o que foi realizado.

DataBase:

Classe responsável pela interface com a base de dados do programa.

***PrivateChat*:**

Classe serializável responsável por guardar os dados referentes a qualquer canal privado (mensagens diretas), recebe o seu UserProfile do seu criador e o UserProfile do user para quem o criador quer mandar mensagem, guarda os dois utilizadores num HashSet de UserProfile e guarda as mensagens recebidas num HashMap com chave Integer e com o value de um objeto Message.

***UserMessages*:**

Classe serializável responsável por guardar os dados que vão ser enviados do utilizador para o servidor para pedir ações deste último, estes pedidos são enviados por um enum de MessageType, e a informação é enviada pelas restantes variáveis do Objeto.

UserProfiles:

Classe serializable responsável por guardar os dados do utilizador, é inicializada com o nome do utilizador, o username, a password e o seu código de login, para além disso guarda num HashSet de String todos os ficheiros a que o utilizador tem acesso e guarda num boolean se o utilizador está conectado.

***UserThreads*:**

Classe responsável por guardar as threads que estão a correr no lado do utilizador, guarda um objeto de cada uma das threads UserHandlerThread, UserNotificationThread e UserFileThread.

Const:

Classe abstrata que contém todas as constantes do programa. Esta classe inclui também as enums usados para troca de mensagens em multicast e entre User e servidor.

Utils:

Esta classe é abstrata e tem o propósito de consolidar os métodos de utilidade. No final acabou por ser apenas um método, que faz o parsing de string para que estas possam ser interpretadas como linhas de comandos.

Server Threads

***HeartbeatThread*:**

Thread responsável por emitir um ping a todos os seus users

MulticastThread:

Thread responsável por estar à escuta de multicasts no servidor. Responde também a vários pedidos que são feitos ao servidor. Esta também envia um update para o outro servidor quando este se conecta sem nada.

ServerThread:

Thread responsável pela inicial receção de informação, seja do cliente ou do servidor que pode também mandar informação. É a sua única função.

UserHandlerThread:

Esta thread é responsável por toda a comunicação entre o servidor e o User após a sua socket TCP ser estabelecida. Cria o UserProfile que vai identificar o User no servidor e faz a gestão de canais e mensagens privadas.

UserNotificationThread;

Esta thread é usada para notificar assincronamente o User. Envia um objeto especialmente criado para a função chamado Notification e fá-lo por uma socket diferente da usada pelo User em comunicação normal com o servidor.

User Threads

NotificationThread:

Thread de notificações do User. Recebe as mensagens do chat e apresenta-as na classe User. Contém uma socket diferente apenas usada para envio de notificações Servidor - User.

StethoscopeThread:

Thread responsável por ouvir os pings do servidor. Caso estes pings parem de ser recebidos a thread avisa o User para este iniciar o processo de reconexão.

2 - Documentação das Decisões

Durante a realização do trabalho, foi necessário a tomada de certas decisões, de modo poder prosseguir com guidelines definidas para como seria o resto do desenvolvimento.

O user verifica se um servidor está vivo através de um timeout de resposta que permite que o servidor responda. Não sendo a forma mais ideal, é versátil e pode ser utilizada para outras funcionalidades como de heartbeat como é o caso.

Quando se pede a capacidade dos outros servidores para confirmar o melhor para o user, é feito um pequeno wait à espera que todos os servidores tenham tempo de fazer multicast da sua capacidade. Isto foi feito porque em certos testes a informação não chegava a tempo.

Para que o servidor tenha uma porta única de heartbeat, ele escreve num ficheiro que posteriormente é aberto por futuros servidores para que saibam qual será o seu heartbeat port. Assim que o servidor decide o seu porto, escreve o novamente no ficheiro.

Para passar pedidos do User para o servidor é usado na maioria uma classe criada para o efeito chamada UserMessages. Está tem vários parâmetros e contrutores que encapsulam possíveis mudanças ou informações que o User queira transmitir.

Para que o user pudesse receber sem preocupações as notificações, foi criada uma socket à parte, ou seja, existem duas sockets por cada user e servidor. Esta thread funciona com recurso a um objeto Notification especialmente criado para o efeito. Tem um código de notificação e informação que se pode anexar à mesma.

Por razões de instabilidade da base de dados, foi tomada a decisão de remover essa funcionalidade. O método de desenvolvimento encorajou a segmentação das diferentes partes do trabalho. Devido a um atraso, a junção e posterior teste da integridade dos dados com a base de dados foi limitada, não permitindo que fosse entregue nesse estado. No entanto será ainda enviado o ficheiro de base de dados apesar de este não ser acedido.

Para que o programa possa ser executado na mesma, todas as variáveis são armazenadas localmente em cada servidor. Esta informação está atualizada entre servidores através de multicast.

3 - Funcionalidades Por Implementar

As funcionalidades que infelizmente ficaram por implementar durante a realização do trabalho foram as seguintes:

- Envio e receção de ficheiros.
- Server ping entre servidores.
- Base de dados. A integridade do programa foi assegurado com dados locais e todo o multicast e atualização é ainda assim feito.

4 - Conclusão

Com os conceitos estudos durante as aulas da unidade curricular de Programação Distribuída e a realização desta primeira meta do trabalho prático, foi concluído que no desenvolvimento de um programa como estes a estruturação e organização da troca de mensagens entre os dois processos é essencial. Várias vezes durante o desenvolvimento foram sofridos atrasos derivados a estas pequenas inconveniências.

Infelizmente a base de dados ficou inoperacional, com problema de troca de informação com os server. O conceito de base de dados, não sendo abordado nas aulas da unidade curricular em si, foi a parte que mais revelou dificuldades, que não se conseguiram ultrapassar.

Ainda assim, a experiência de desenvolvimento foi bastante enriquecedora e elucidativa dos conceitos lecionados nas aulas. Todos os conhecimentos foram postos respetivamente à prova e apesar de incompletos, o código está relativamente bem organizado e robusto.

Por último, conclui-se que, tal como em outras linguagens orientadas a objetos, a existência dos mesmos durante a programação é algo que torna a programação extremamente mais intuitiva.